Forty Years of Text Indexing

Alberto Apostolico, Maxime Crochemore, Martin Farach-Colton, Zvi Galil, S. Muthukrishnan

Forty Years of Text Indexing

Alberto Apostolico*, Maxime Crochemore**, Martin Farach-Colton***, Zvi Galil†, and S. Muthukrishnan ‡

Abstract. This paper reviews the first 40 years in the life of textual inverted indexes, their many incarnations, and their applications. The paper is non-technical and assumes some familiarity with the structures and constructions discussed. It is not meant to be exhaustive. It is meant to be a tribute to a ubiquitous tool of string matching — the suffix tree and its variants — and one of the most persistent subjects of study in the theory of algorithms.

Keywords: pattern matching, string searching, bi-tree, suffix tree, dawg, suffix automaton, factor automaton, suffix array, FM-index, wavelet tree.

1 Prolog

When William Legrand finally decrypted the string:

> 53‡‡†305))6*,48264‡.)4‡);806",48†8P60))85;1‡(;:‡*8†83(88)5*†,46(;88*96 *?;8)*‡(;485);5*†2:*‡(;4956*2(5*4)8P8*;4069285);)6‡8)4‡‡;1(‡9;48081;8:8 ‡1;48 85;4)485†528806*81(ddag9;48;(88;4(‡?34;48)4‡;161;:188;‡?;

it did not seem to make much more sense than it did before. The decoded message read: "A good glass in the bishop's hostel in the devil's seat forty-one degrees and thirteen minutes northeast and by north main branch seventh limb east side shoot from the left eye of the death's-head a bee line from the tree through the shot fifty feet out." But at least it did sound more like natural language, and eventually guided the main character of Edgar Allan Poe's "The Gold Bug" [56] to discover the treasure he had been after. Legrand solved a substitution cipher using symbol frequencies. He first looked for the most frequent symbol and changed it into the most frequent letter of English, then similarly treated the second most frequent symbol, and so on.

* College of Computing, Georgia Institute of Technology, 801 Atlantic Drive, Atlanta, GA 30318, USA axa@cc.gatech.edu

** King's College London, London WC2R 2LS, UK and Université Paris-Est, France maxime.crochemore@kcl.ac.uk

*** Department of Computer Science, Rutgers University, Piscataway, NJ 08854, USA farach@cs.rutgers.edu

† College of Computing, Georgia Institute of Technology, 801 Atlantic Drive, Atlanta, GA 30318, USA galil@cc.gatech.edu

‡ Department of Computer Science, Rutgers University, Piscataway, NJ 08854, USA muthu@cs.rutgers.edu

Both before and after 1843, the natural impulse when faced with some mysterious message has been to count frequencies of individual tokens or subassemblies in search of a clue. Perhaps one of the most intense and fascinating subjects for this kind of scrutiny has been bio-sequences. As soon as some such sequences became available, statistical analyses tried to link characters or blocks of characters to relevant biological function. With the early examples of whole genomes emerging in the mid 90's, it seemed natural to count the occurrences of all blocks of size 1, 2, etc. up to any desired length, looking for statistical characterizations of coding regions, promoter regions, etc. [67].

This review is not about cryptography. It is about a data structure and its variants, and the many surprising and useful features it carries. Among these is the fact that, to set up a statistical table of occurrences for *all* substrings, of *any* length, of a text string of n characters, it only takes time and space linear in the length of the text string. While nobody would be so foolish to solve the problem by generating all exponentially many possible substrings and then counting their occurrences one by one, a text string may still contain $O(n^2)$ distinct substrings, so that tabulating all of them in linear space, never mind linear time, seems already puzzling.

Over the years, such structures have held center stage in text searching, indexing, statistics, and compression as well as in the assembly, alignment and comparison of biosequences. Their range of scope extends to areas as diverse as detecting plagiarism, finding surprising substrings in a text, testing the unique decipherability of a code, and more. Their impact on Computer Science and IT at large cannot be overstated. Text and Web searching and Bioinformatics would not be the same without them. In 2013, the Combinatorial Pattern Matching symposium celebrates the 40th anniversary of the appearance of Wiener's paper with a special session entirely dedicated to that event.

2 History Bits and Pieces

At the dawn of "stringology", Don Knuth conjectured that the problem of finding the longest substring common to two long text sequences of total length n required $\Omega(n \log n)$ time. An $O(n \log n)$-time had been provided by Karp, Miller and Rosenberg [40]. That construction was destined to play a role in parallel pattern matching [6, 24, 31, 32], but Knuth's conjecture was short lived: in 1973, Peter Weiner showed that the problem admitted an elegant linear-time solution [68], as long as the alphabet of the string was fixed. Such a solution was actually a byproduct of a construction he had originally set up for a different purpose, i.e., identifying any substring of a textfile without specifying all of them. In doing so, Weiner introduced a notion of a textual inverted index that would elicit refinements, analyses and applications for forty years and counting, a feature hardly shared by any other data structure.

Weiner's original construction processed the textfile from right to left. As each new character was read in, the structure, which he called a "bi-tree", would be updated to accommodate longer and longer suffixes of the textfile. Thus this

was an inherently offline construction, since the text had to be known in its entirety before the construction could begin. Alternatively, one could say that the algorithm would build online the structure for the reverse of the text. About three years later, Ed McCreight provided a left-to-right algorithm and changed the name of the structure to "suffix tree", a name that would stick [52]. In unpublished lecture notes of 1975, Vaughan Pratt displayed the duality of this structure and Weiner's "repetition finder" [57]. McCreight's algorithm was still inherently offline, and it immediately triggered a craving for an online version. Some partial attempts at an on-line algorithm were performed [42, 49], but such a variant had to wait almost two decades for Esko Ukkonen's paper in 1995 [65]. In all linear constructions, linearity was based on the assumption of a finite alphabet and took $O(n \log n)$ time in general. In 1997, Martin Farach introduced an algorithm that abandoned the one-suffix-at-time approach prevalent until then; this algorithm gives a linear-time reduction from suffix-tree construction to character sorting, and thus runs in linear time, for example, even when the alphabet is of size polynomial in the input size [26].

Around 1984, Anselm Blumer, et al. [12–14] and Maxime Crochemore [19] exposed the surprising result that the smallest finite automaton recognizing all and only the suffixes of a string of n characters has only $O(n)$ states and edges. Initially coined as a directed acyclic word graph (DAWG), it can even be reduced if all states are terminal states. It then accepts all subsstrings of the string and is called the factor/substring automaton. Although it has never been fully elucidated, it seems that Anatoli Slissenko [59, 60] ended up with a similar structure for his work on the detection of repetitions in strings. These automata provided another more efficient counterexample to Knuth's conjecture when they are used, against the grain, as pattern matching machines (see [20]).

The appearance of suffix trees dovetailed with some interesting and independent developments in information theory. In his famous approach to the notion of information, Kolmogorov [45] equated the information or structure in a string to the length of the shortest program that would be needed to produce that string by a Universal Turing Machine. The unfortunate thing is that this measure is not computable and even if it were, most long strings would be incompressible (would lack a short program producing them), since there are increasingly many long strings and comparatively much fewer short programs (themselves strings).

The regularities exploited by Kolmogorov's universal and omniscient machine could be of any conceivable kind, but what if one limited them to the syntactic redundancies affecting a text in form of repeated substrings? If a string is repeated many times one could profitably encode all occurrences by a pointer to a common copy. This copy could be internal or external to the text. In the latter case one could have pointers going in both directions or only in one, allow or forbid nesting of pointers, etc. In his doctoral thesis, Jim Storer [61–63] showed that virtually all such "macro schemes" are intractable, *except one*. Not long before that, in a landmark paper entitled "On the Complexity of Finite Sequences" [48], Abraham Lempel and Jacob Ziv had proposed a variable-to-block encoding based on a simple parsing of the text and with the feature that the

compression achieved would match in the limit that produced by a compressor tailored to the source probabilities. Thus, by a remarkable alignment of stars, the compression method brought about by Lempel and Ziv was not only optimal in the information theoretic sense; it found an optimal, linear-time implementation by the suffix tree, as was detailed immediately by Michael Rodeh, Vaugham Pratt and Simon Even [58].

In his original paper, Weiner listed a few applications of his "bi-tree", including most notably off-line string searching: preprocessing a text file to support queries that return the occurrences of a given pattern in time linear in the length of the pattern. And of course, the "bi-tree" addressed Knuth's conjecture, by showing how to find a longest substring common to two files in linear time for finite alphabet. There followed unpublished notes by Pratt entitled "Improvements and Applications for the Weiner Repetition Finder" [57]. A decade later, Alberto Apostolico would list more applications in a paper entitled "The Myriad Virtues of Suffix Trees" [3], and two decades later suffix trees and companion structures elicited with their applications several chapters in reference books by Crochemore and Rytter [25], Dan Gusfield [35], and Crochemore, Hancart, Lecroq [21].

The space required by suffix trees has been a nuisance in applications where they were needed the most. With genomes in the order of gigabytes, for instance, it makes a big difference to need space 20 times bigger than the source versus, say, only 11 times that big. For a few lusters, Stefan Kurtz and his co-workers devoted their effort to cleverly allocating the tree and some of its companion structures [46]. In 2001 David R. Clark, J. Ian Munro proposed one of the best space-saving methods on secondary storage [18]. Clark and Munro's "succinct suffix tree" sought to preserve as much of the structure of the suffix tree as possible. Udi Manber and Eugene W. Myers took a different approach, however. In 1990 [50, 51], they introduced the "suffix array," which eliminated most of the structure of the suffix tree, but was still able to implement many of the same operations, at a cost of only twice the input size. Although the suffix array seemed at first to be a different data structure than the suffix tree, over time they have come to be more and more similar. For example, Manber and Myers's original construction of the suffix array took $O(n \log n)$ time for any alphabet, but the suffix array could be constructed in linear time from the suffix tree for any alphabet. In 2001, Toru Kasai et al. [41] showed that the suffix tree could be constructed in linear time from the suffix array. The suffix array was shown to be a succinct representation of the suffix tree. In 2003, three groups [39, 43, 44] modified in three different ways Farach's algorithm for suffix tree construction to give the first linear-time algorithms for directly constructing the suffix array, that is, the first linear-time algorithms for computing suffix arrays that did not first compute the full suffix tree. With fast construction algorithms and small space, the suffix array is the suffix-tree variant that has gained the most widespread adoption in software systems.

Actually, the history of inverted indexes and compression is tightly intertwined. This should not come as a surprise, since the redundancies that pattern

discovery tries to unearth are ideal candidates to be removed for purposes of compression. In 1994, M. Burrows and D. J. Wheeler proposed a puzzling data compression method in a report that "as it happens sometimes to results that are just too simple and elegant" ended it up never finding an archival venue [16]. Circa 1995, Amihood Amir, Gary Benson and Martin Farach posed the problem of searching in compressed texts [1, 2]. In 2000, Paolo Ferragina and Giovanni Manzini introduced a compressed full text index based on the Burrows-Wheeler transform [29, 30]. In the same year, Roberto Grossi and Jeffrey Scott Vitter presented compressed versions of suffix trees and suffix arrays [34]. These structures supported searching without decompression while being possibly smaller than the source file. This was extended to compressed tree indexing problems in [28] using a modification of the Burrows-Wheeler transform.

3 Fallout, Extensions and Challenges

As highlighted in our prolog, there has been hardly any application of text processing that did not need these indexes at one point or another. A prominent case has been searching with errors, a problem first efficiently tackled in 1985 by Gad Landau in his PhD thesis [47]. In this kind of searches, one looks for substrings of the text that differ from the pattern in a limited number of errors such as a single character deletion, insertion or substitution. To efficiently solve this problem, Landau combined Suffix Trees with a clever solution to the so-called lowest common ancestor (LCA) problem. The LCA problem assumes that a rooted tree is given and then for any pair of nodes, it seeks the lowest node in the tree that is an ancestor of both [37] (see [11] and references therein for subsequent, simpler constructions). It is seen that following a linear time preprocessing of the tree any LCA query can be answered in constant time. Landau used LCA queries on Suffix Trees to perform contant-time jumps over segments of the text that would be guaranteed to match the pattern. When k errors are allowed, the search for an occurrence at any given position can be abandoned after k such jumps. This leads to an algorithm that searches for a pattern with k errors in a text of n characters in $O(nk)$ steps.

Among the basic primitives supported by suffix trees and arrays one finds of course searching for a pattern in a text in time proportional to the length of the pattern rather than the text. In fact, it is even possible to enumerate occurrences in time proportional to their number and, with trivial preprocessing of the tree, tell the total number of occurrences for any query pattern in time proportional to the pattern size. The problem of finding the longest substring appearing twice in a text or shared between two files has been already mentioned: this is probably where it all started. A germane problem is that of detecting squares, repetitions and maximal periodicities in a text, a problem rooted in work by Axel Thue dated more than a century ago [64], a problem with multiple contemporary applications in compression and DNA analysis. A square is a pattern consisting of two consecutive occurrences of the same string. Suffix trees have been used to detect in optimal $O(n \log n)$ time all squares (or repetitions) in a text, each with

its set of starting positions [7], and later to find and store all distinct square substrings in a text in linear time [36]. Squares play a role in an augmentation of the suffix tree suitable to report, for any query pattern, the number of its non-overlapping occurrences [8, 15].

There are multiple uses of suffix trees in setting up some kind of signature for texstrings, as well as measures of similarity or difference. Among the latter, there is the problem of computing the forbidden or absent words of a text, which are strings that do not appear in the text while all of their substrings do [10, 22]. Such words subtend, among other things, to an original approach to text compression [23]. Once regarded as the succinct representation of the "bag-of-words" of a text, suffix trees can be used to assess the similarity of two textfiles, thereby supporting clustering, document classification and even phylogeny [5, 17, 66]. Intuitively, this is done by assessing how much the trees relative the two input sequences have in common. Suitably enriched with the probability affecting the substring ending at each node, a tree can be used to detect surprisingly over-represented substrings of any length [4], e.g., in the quest of promoter regions in biosequences [67].

The suffix tree of the concatenation of say, $k \geq 2$ texfiles, supports efficient solutions to problems arising in domains ranging from plagiarism detection to motif discovery in biosequences. The need for k distinct end-markers poses some subtleties in maintaining linear time, for which the reader is referred to [35]. In its original form, the problem was called "color problem" and seeks to report, for any given query string and in time linear in the query, how many documents out of the total of k contain each at least one occurrence of the query. A simple and elegant solution was given in 1992 by Lucas C. K. Hui [38]. More recently, it was extended to a variety of document listing problems, where once a set of text documents are preprocessed, one can return the list of all documents that contain a query pattern in time proportional to the number of such documents, not the total number of occurrences [53].

One surprising variant of the suffix tree was introduced by Brenda Baker for purposes of detection of plagiarism in student reports as well as optimization in software development [9] . This variant of pattern matching, called "parameterized matching", enables to find program segments that are identical up to a systematic change of parameters, or substrings that are identical up to a systematic relabeling or permutation of the characters in the alphabet.

One obvious extension of the notion of a suffix tree is to more than one dimension, albeit the mechanics of the extension itself is far from obvious [54,55] Among more distant relatives, one finds "wavelet trees". Originally proposed as a representation of compressed suffix arrays [33], wavelet trees enable to perform on general alphabets the ranking and selection primitives previously limited to bit vectors, and more [27].

The list could go on and on, but the scope of this paper was not meant be exhaustive. Actually, after forty years of unrelenting developments, it is fair to assume that the list will continue to grow. On the other hand, many of the observed sequences are expressed in numbers rather than characters, and in

both cases are affected by various types of errors. While the outcome of a two character comparison is just one bit, two numbers can be more or less close, depending on their difference or some other metric. Likewise, two textstrings can be more or less similar, depending on the number of elementary steps necessary to change one in the other. The most disruptive aspect of this framework is the loss of the transitivity property that subtends to the most efficient exact string matching solutions. And yet indexes capable of supporting fast and elegant approximate pattern queries of the kind just highlighted would be immensely useful. Hopefully, they will come up soon and, in time, get their own 40th anniversary celebration.

References

1. A. Amir, G. Benson, and M. Farach. Let sleeping files lie: Pattern matching in Z-compressed files. In *Proceedings of the 5th ACM-SIAM Annual Symposium on Discrete Algorithms*, pages 705–714, Arlington, VA, 1994.
2. A. Amir, G. Benson, and M. Farach. Let sleeping files lie: Pattern matching in Z-compressed files. *J. Comput. Syst. Sci.*, 52(2):299–307, 1996.
3. A. Apostolico. The myriad virtues of suffix trees. In A. Apostolico and Z. Galil, editors, *Combinatorial Algorithms on Words*, volume 12 of *NATO Advanced Science Institutes, Series F*, pages 85–96. Springer-Verlag, Berlin, 1985.
4. A. Apostolico, M. E. Bock, and S. Lonardi. Monotony of surprise and large-scale quest for unusual words. *Journal of Computational Biology*, 10(3/4):283–311, 2003.
5. A. Apostolico, O. Denas, and A. Dress. Efficient tools for comparative substring analysis. *Journal of Biotechnology*, 149(3):120–126, 2010.
6. A. Apostolico, C. Iliopoulos, G. M. Landau, B. Schieber, and U. Vishkin. Parallel construction of a suffix tree with applications. *Algorithmica*, 3:347–365, 1988.
7. A. Apostolico and F. P. Preparata. Optimal off-line detection of repetitions in a string. *Theor. Comput. Sci.*, 22(3):297–315, 1983.
8. A. Apostolico and F. P. Preparata. Data structures and algorithms for the strings statistics problem. *Algorithmica*, 15(5):481–494, May 1996.
9. B. S. Baker. Parameterized duplication in strings: Algorithms and an application to software maintenance. *SIAM J. Comput.*, 26(5):1343–1362, 1997.
10. M.-P. Béal, F. Mignosi, and A. Restivo. Minimal forbidden words and symbolic dynamics. In *STACS 96, 13th Annual Symposium on Theoretical Aspects of Computer Science, Grenoble, France, February 22-24, 1996, Proceedings*, volume 1046 of *Lecture Notes in Computer Science*, pages 555–566. Springer, 1996.
11. M. A. Bender and M. Farach-Colton. The lca problem revisited. In *LATIN 2000: Theoretical Informatics, 4th Latin American Symposium, Punta del Este, Uruguay, April 10-14, 2000, Proceedings*, volume 1776 of *Lecture Notes in Computer Science*, pages 88–94. Springer, 2000.
12. A. Blumer, J. Blumer, A. Ehrenfeucht, D. Haussler, M. T. Chen, and J. Seiferas. The smallest automaton recognizing the subwords of a text. *Theor. Comput. Sci.*, 40(1):31–55, 1985.
13. A. Blumer, J. Blumer, A. Ehrenfeucht, D. Haussler, and R. McConnell. Building a complete inverted file for a set of text files in linear time. In *Proceedings of the 16th ACM Symposium on the Theory of Computing*, pages 349–351, Washington, D.C., 1984. ACM Press.

14. A. Blumer, J. Blumer, A. Ehrenfeucht, D. Haussler, and R. Mcconnell. Complete inverted files for efficient text retrieval and analysis. *J. Assoc. Comput. Mach.*, 34(3):578–595, 1987.
15. G. S. Brodal, R. B. Lyngsø, A. Östlin, and C. N. S. Pedersen. Solving the string statistics problem in time o(n log n). In *Automata, Languages and Programming, 29th International Colloquium, ICALP 2002, Malaga, Spain, July 8-13, 2002, Proceedings*, volume 2380 of *Lecture Notes in Computer Science*, pages 728–739. Springer, 2002.
16. M. Burrows and D. J. Wheeler. A block-sorting lossless data compression algorithm. Technical Report 124, Digital Equipments Corporation, May 1994.
17. S. Chairungsee and M. Crochemore. Using minimal absent words to build phylogeny. *Theoretical Computer Science*, 450(1):109–116, 2012.
18. D. R. Clark and J. I. Munro. Efficient suffix trees on secondary storage. In *Proceedings of the 7th ACM-SIAM Annual Symposium on Discrete Algorithms*, pages 383–391, Atlanta, Georgia, 1996.
19. M. Crochemore. Transducers and repetitions. *Theor. Comput. Sci.*, 45(1):63–86, 1986.
20. M. Crochemore. Longest common factor of two words. In Ehrig, Kowalski, Levi, and Montanari, editors, *TAPSOFT'87 (Pisa, 1987)*, number 249 in LNCS, pages 26–36. Springer-Verlag, 1987.
21. M. Crochemore, C. Hancart, and T. Lecroq. *Algorithms on Strings*. Cambridge University Press, Cambridge, 2007.
22. M. Crochemore, F. Mignosi, and A. Restivo. Automata and forbidden words. *Information Processing Letters*, 67(3):111–117, 1998.
23. M. Crochemore, F. Mignosi, A. Restivo, and S. Salemi. Data compression using antidictonaries. *Proceedings of the I.E.E.E.*, 88(11):1756–1768, 2000. Special issue *Lossless data compression* edited by J. Storer.
24. M. Crochemore and W. Rytter. Usefulness of the Karp-Miller-Rosenberg algorithm in parallel computations on strings and arrays. *Theor. Comput. Sci.*, 88(1):59–82, 1991.
25. M. Crochemore and W. Rytter. *Text algorithms*. Oxford University Press, 1994.
26. M. Farach. Optimal suffix tree construction with large alphabets. In *Proceedings of the 38th IEEE Annual Symposium on Foundations of Computer Science*, pages 137–143, Miami Beach, FL, 1997.
27. P. Ferragina, R. Giancarlo, and G. Manzini. The myriad virtues of wavelet trees. *Inf. Comput.*, 207(8):849–866, 2009.
28. P. Ferragina, F. Luccio, G. Manzini, and S. Muthukrishnan. Compressing and indexing labeled trees, with applications. *J. ACM*, 57(1), 2009.
29. P. Ferragina and G. Manzini. Opportunistic data structures with applications. In *FOCS*, pages 390–398, 2000.
30. P. Ferragina and G. Manzini. Indexing compressed text. *J. ACM*, 52(4):552–581, 2005.
31. Z. Galil. Optimal parallel algorithms for string matching. In *Proceedings of the 16th ACM Symposium on the Theory of Computing*, pages 240–248, Washington, D.C., 1984. ACM Press.
32. Z. Galil. Optimal parallel algorithms for string matching. *Inf. Control*, 67(1–3):144–157, 1985.
33. R. Grossi, A. Gupta, and J. S. Vitter. High-order entropy-compressed text indexes. In *SODA*, pages 841–850, 2003.

34. R. Grossi and J. S. Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. In *Proceedings ACM Symposium on the Theory of Computing*, pages 397–406, Portland, Oregon, 2000. ACM Press.
35. D. Gusfield. *Algorithms on strings, trees and sequences: computer science and computational biology.* Cambridge University Press, Cambridge, 1997.
36. D. Gusfield and J. Stoye. Linear time algorithms for finding and representing all the tandem repeats in a string. *J. Comput. Syst. Sci.*, 69(4):525–546, 2004.
37. D. Harel and R. E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.*, 13(2):338–355, 1984.
38. L. C. K. Hui. Color set size problem with applications to string matching. In A. Apostolico, M. Crochemore, Z. Galil, and U. Manber, editors, *Proceedings of the 3rd Annual Symposium on Combinatorial Pattern Matching*, number 644 in Lecture Notes in Computer Science, pages 230–243, Tucson, AZ, 1992. Springer-Verlag, Berlin.
39. J. Kärkkäinen and P. Sanders. Simple linear work suffix array construction. In J. C. M. Baeten, J. K. Lenstra, J. Parrow, and G. J. Woeginger, editors, *Automata, Languages and Programming, 30th International Colloquium, ICALP 2003, Eindhoven, The Netherlands, June 30 - July 4, 2003. Proceedings*, volume 2719 of *Lecture Notes in Computer Science*, pages 943–955, 2003.
40. R. M. Karp, R. E. Miller, and A. L. Rosenberg. Rapid identification of repeated patterns in strings, trees and arrays. In *Proceedings of the 4th ACM Symposium on the Theory of Computing*, pages 125–136, Denver, CO, 1972. ACM Press.
41. T. Kasai, G. Lee, H. Arimura, S. Arikawa, and K. Park. Linear-time longest-common-prefix computation in suffix arrays and its applications. In *CPM*, pages 181–192. Springer-Verlag, 2001.
42. M. Kempf, R. Bayer, and U. Güntzer. Time optimal left to right construction of position trees. *Acta Inform.*, 24(4):461–474, 1987.
43. D. K. Kim, J. S. Sim, H. Park, and K. Park. Constructing suffix arrays in linear time. *J. Discrete Algorithms*, 3(2-4):126–142, 2005.
44. P. Ko and S. Aluru. Space efficient linear time construction of suffix arrays. *J. Discrete Algorithms*, 3(2-4):143–156, 2005.
45. A. N. Kolmogorov. Three approaches to the quantitative definition of information. *Problems of Information Transmission*, 1(1):1–7, 1965.
46. S. Kurtz. Reducing the space requirements of suffix trees. *Softw. Pract. Exp.*, 29(13):1149–1171, 1999.
47. G. M. Landau. *String matching in erroneus input.* Ph. D. Thesis, Department of Computer Science, Tel-Aviv University, 1986.
48. A. Lempel and J. Ziv. On the complexity of finite sequences. *IEEE Trans. Inf. Theory*, 22:75–81, 1976.
49. M. E. Majster and A. Ryser. Efficient on-line construction and correction of position trees. *SIAM J. Comput.*, 9(4):785–807, 1980.
50. U. Manber and G. Myers. Suffix arrays: a new method for on-line string searches. In *Proceedings of the 1st ACM-SIAM Annual Symposium on Discrete Algorithms*, pages 319–327, San Francisco, CA, 1990.
51. U. Manber and G. Myers. Suffix arrays: a new method for on-line string searches. *SIAM J. Comput.*, 22(5):935–948, 1993.
52. E. M. McCreight. A space-economical suffix tree construction algorithm. *J. Algorithms*, 23(2):262–272, 1976.
53. S. Muthukrishnan. Efficient algorithms for document listing problems. In *Proceedings of the 13th ACM-SIAM Annual Symposium on Discrete Algorithms*, pages 657–666, 2002.

54. J. C. Na, P. Ferragina, R. Giancarlo, and K. Park. Two-dimensional pattern indexing. In *Encyclopedia of Algorithms*. 2008.
55. J. C. Na, R. Giancarlo, and K. Park. On-line construction of two-dimensional suffix trees in o(n^2 log n) time. *Algorithmica*, 48(2):173–186, 2007.
56. E. A. Poe. *The Gold-Bug and Other Tales*. Dover Thrift Editions Series. Dover, 1991.
57. V. Pratt. Improvements and applications for the Weiner repetition finder. Manuscript, 1975.
58. M. Rodeh, V. Pratt, and S. Even. Linear algorithm for data compression via string matching. *J. Assoc. Comput. Mach.*, 28(1):16–24, 1991.
59. A. O. Slisenko. Determination in real time of all the periodicities in a word. *Sov. Math. Dokl.*, 21:392–295, 1980.
60. A. O. Slisenko. Detection of periodicities and string matching in real time. *J. Sov. Math.*, 22:1316–1386, 1983.
61. J. A. Storer. NP-completeness results concerning data compression. Report 234, Princeton University, 1977.
62. J. A. Storer and T. G. Szymanski. The macro model for data compression. In *Proceedings of the 10th ACM Symposium on the Theory of Computing*, pages 30–39, San Diego, CA, 1978. ACM Press.
63. J. A. Storer and T. G. Szymanski. Data compression via textual substitution. *J. Assoc. Comput. Mach.*, 29(4):928–951, 1982.
64. A. Thue. Über die gegenseitige lage gleicher teile gewisser zeichenreichen. *Nor. Vidensk. Selsk. Skr. Mat. Nat. Kl.*, 1:1–67, 1912.
65. E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.
66. I. Ulitsky, D. Burstein, T. Tuller, and B. Chor. The average common substring approach to phylogenomic reconstruction. *Journal of Computational Biology*, 13(2):336–350, 2006.
67. J. van Helden, B. André, and J. Collado-Vides. Extracting regulatory sites from the upstream region of the yeast genes by computational analysis of oligonucleotides. *J. Mol. Biol.*, 281:827–842, 1998.
68. P. Weiner. Linear pattern matching algorithm. In *Proceedings of the 14th Annual IEEE Symposium on Switching and Automata Theory*, pages 1–11, Washington, DC, 1973.